

**Apostila de Introdução a Lógica de Programação
Usando Visualg
por**

MsC. Jaqueline Souza de Oliveira Valladares



Caros Alunos,

Essa apostila de Introdução a Lógica de programação é parte da disciplina Introdução a Informática destinada aos alunos do Primeiro Ano do Curso Técnico em Informática do IFBA, Campus Santo Amaro.

O objetivo é facilitar o aprendizado de introdução a lógica de programação de forma lúdica e interativa. Inicializando conceitos básicos de lógica de programação e a estrutura linear/sequencial de um programa simples.

Para ilustrar os conceitos referentes a introdução a lógica de programação será utilizado o programa Visualg 2.0.

Um programa ideal para estudantes por ser simples e bastante intuitivo. Visualg não depende de componentes como DLLs e OCXs. Ao ser instalado, seus arquivos não são copiados para nenhuma outra pasta que não seja a que ele tenha sido instalado. Exige aproximadamente 1Mb de espaço em disco e pode ser executado em sistemas operacionais Windows 95 ou superiores.

A versão 2.0 conta com:

- Criação de subprogramas (com procedimentos e funções)
- Criação de variáveis locais
- Passagem de parâmetros por referência
- Envio de algoritmo por e-mail, diretamente do programa
- Conversão automática de algoritmos para TurboPascal

O programa pode ser encontrado na seguinte URL para fazer o download gratuitamente:
<http://www.baixaki.com.br/download/Visualg.htm>

Espero que possam interagir e solucionar os exercícios propostos na apostila. Bom Estudo!!

Jaqueline Oliveira

Introdução a Lógica de Programação

Para que serve a Lógica de Programação?

Necessária para pessoas que desejem trabalhar com desenvolvimento de sistemas e programas, ela permite definir a sequência lógica para o seu desenvolvimento.

É a técnica de encadear pensamentos, através de uma **seqüência lógica**, para atingir um determinado objetivo de um sistema. Estes pensamentos podem ser descritos como uma seqüência de **instruções**, que devem ser seguidas para se cumprir uma determinada tarefa.

Mas o que vem a ser uma Seqüência Lógica?

São os passos executados até atingir um objetivo ou solução de um problema. Convém ressaltar que uma ordem isolada não permite realizar o processo completo, para isso é necessário um conjunto de **instruções** colocadas em ordem seqüencial lógica.

Atenção:

Em informática, é a informação que indica a um computador uma ação elementar a executar. Uma instrução tomada em separado não tem muito sentido; para obtermos o resultado, precisamos colocar em prática o conjunto de todas as instruções, na ordem correta.

Antes de iniciarmos a escrita de um programa é necessário conhecer alguns conceitos básicos:

- Constantes e Variáveis

a) Contante

Segundo Farrer et al (2011, p. 29) “contante é um determinado valor fixo que não se modifica ao longo do tempo, durante a execução de um programa.” Conforme o seu tipo pode ser classificada em: Numérica, Lógica ou Literal.

Por exemplo: O número fixo (2) usado para calcular a média aritmética (Media) é considerado uma constante numérica.

Media <- (N1+N2)/2

As constantes não precisam ser declaradas num programa.

b) Variável

É a representação simbólica dos elementos de um certo conjunto. Cada variável corresponde a uma posição de memória, cujo conteúdo pode se alterado ao longo do tempo durante a execução de um programa.

Segundo Venancio (1997, p.5) “ variáveis de memória são endereços da memória RAM (Randomic Access Memory) do computador, onde são armazenados temporariamente os dados utilizados por um programa durante seu processamento, e que como diz o próprio nome, podem ter seus conteúdos alterados durante o processamento do programa.”

Embora uma variável possa assumir diferentes valores, ela só pode armazenar um valor a cada instante.

As variáveis podem ser de quatro tipos:

Numéricas

Específicas para armazenamento de números, que posteriormente poderão ser utilizados para cálculos

Podem ser ainda classificadas como Inteiras ou Reais

Caracteres

Específicas para armazenamento de conjunto de caracteres que não contenham números (literais)

Exemplo: nomes

Alfanuméricas

Específicas para dados que contenham letras e/ou números

Pode em determinados momentos conter somente dados numéricos ou somente literais

OBS: Se usado somente para armazenamento de números, **não** poderá ser utilizada para operações matemáticas

Lógicas

Armazenam somente dados lógicos que podem ser Verdadeiro ou Falso

O Visualg aceita cinco tipos de variáveis, que são:

TIPO DE VARIÁVEL	DESCRIÇÃO
INTEIRO	define variáveis numéricas do tipo inteiro, ou seja, sem casas decimais.
REAL	define variáveis numéricas do tipo real, ou seja, com casas decimais.
CARACTER	define variáveis do tipo <i>string</i> , ou seja, cadeia de caracteres.
LOGICO	define variáveis do tipo <i>booleano</i> , ou seja, com valor VERDADEIRO ou FALSO.
VETOR	são uma maneira de armazenar vários dados num mesmo nome de variável através do uso de índices numéricos. Declara-se vetores de maneira muito semelhante à declaração de variáveis normais. A única diferença é que depois do nome da variável deve ser informada a quantidade de elementos do vetor.

Por exemplo:

```
var
nome: caracter
sexo: logico
numero: inteiro
raiz: real
A: vetor
```

Figura 1 – Exemplo de declaração de variáveis no Visualg

Nesse exemplo são criadas cinco variáveis: nome, sexo, numero, A e raiz. Todas elas foram declaradas na área destinada a declaração: var. Cada uma delas possui um tipo diferente. Conforme Farrer et all (2011, p.31) “ no momento em que se declara uma variável, é feita a associação do nome escolhido, ou identificador, com a respectiva

posição de memória que o mesmo assa a simbolizar.”

Atenção:

Todas as variáveis a serem utilizadas num programa, precisam serem declaradas, informando o seu tipo. A fim de que o sistema possa alocar na memória seu armazenamento.

Nomes de Variáveis e sua Declaração

Os nomes das variáveis devem começar por uma letra e depois conter letras, números ou *underline*, até um limite de 30 caracteres. As variáveis podem ser simples ou estruturadas (na versão atual, os vetores podem ser de uma ou duas dimensões). Não pode haver duas variáveis com o mesmo nome, com a natural exceção dos elementos de um mesmo vetor.

A seção de declaração de variáveis começa com a palavra-chave *var*, e continua com as seguintes sintaxes:

<lista-de-variáveis> : *<tipo-de-dado>*
<lista-de-variáveis> : vetor "["*<lista-de-intervalos>*"]" de *<tipo-de-dado>*

Na *<lista-de-variáveis>*, os nomes das variáveis estão separados por vírgulas. Na *<lista-de-intervalos>*, os *<intervalo>* são separados por vírgulas, e têm a seguinte sintaxe:

<intervalo>: *<valor-inicial>* .. *<valor-final>*

Na versão atual do VisuAlg, tanto *<valor-inicial>* como *<valor-final>* devem ser inteiros. Além disso, exige-se evidentemente que *<valor-final>* seja maior do que *<valor-inicial>*.

Exemplos:

```
var a: inteiro
    Valor1, Valor2: real
    vet: vetor [1..10] de real
    matriz: vetor [0..4,8..10] de inteiro
    nome_do_aluno: caractere
    sinalizador: logico
```

Note que não há a necessidade de ponto e vírgula após cada declaração: basta pular linha. A declaração de vetores é análoga à linguagem Pascal: a variável *vet* acima tem 10 elementos, com os índices de [1] a [10], enquanto *matriz* corresponde a 15 elementos com índices [0,8], [0,9], [0,10], [1,8], [1,9], [1,10], ... até [4,10]. O número total de variáveis suportado pelo VisuAlg é 500 (cada elemento de um vetor é contado individualmente).

- Operadores

Meios pelo qual incrementamos, decrementamos, comparamos e avaliamos dados dentro do computador.

Tipos de operadores:

- Operadores Aritméticos : Utilizados para obter resultados numéricos

As principais operações são: Adição, Subtração, Multiplicação, Divisão e Exponenciação.

Operadores	Descrição
+, -	Operadores unários, isto é, são aplicados a um único operando. São os operadores aritméticos de maior precedência. Exemplos: -3, +x. Enquanto o operador unário - inverte o sinal do seu operando, o operador + não altera o valor em nada o seu valor.
\	Operador de divisão inteira. Por exemplo, $5 \setminus 2 = 2$. Tem a mesma precedência do operador de divisão tradicional
. +, -, *, /	Operadores aritméticos tradicionais de adição, subtração, multiplicação e divisão. Por convenção, * e / têm precedência sobre + e -. Para modificar a ordem de avaliação das operações, é necessário usar parênteses como em qualquer expressão aritmética
MOD ou %	Operador de módulo (isto é, resto da divisão inteira). Por exemplo, $8 \text{ MOD } 3 = 2$. Tem a mesma precedência do operador de divisão tradicional. Ele segura o resto da divisão.
^	Operador de potenciação. Por exemplo, $5 ^ 2 = 25$. Tem a maior precedência entre os operadores aritméticos binários (aqueles que têm dois operandos).

- Operadores Lógico: Utilizados para combinar resultados de expressões, retornando se o resultado final é verdadeiro ou falso

Os operadores lógicos são:

E / AND

Uma expressão AND (E) é verdadeira se todas as condições forem verdadeiras

OU/OR

Uma expressão OR (OU) é verdadeira se pelo menos uma condição for verdadeira

Nao/NOT

Um expressão NOT (NÃO) inverte o valor da expressão ou condição, se verdadeira inverte para falsa e vice-versa.

- Operadores Relacionais: São utilizados para comparar string de caracteres e números

Os valores a serem comparados podem ser caracteres ou variáveis

Sempre retornam valores lógicos

Verdadeiro/True

Falso/False

Para estabelecer prioridades no que diz respeito a qual operação executar primeiro, utilize os parênteses

Operadores	Descrição
=, <, >, <=, >=, <>	Respectivamente: igual, menor que, maior que, menor ou igual a, maior ou igual a, diferente de. São utilizados em expressões lógicas para se testar a relação entre dois valores do mesmo tipo. Exemplos: 3 = 3 (3 é igual a 3?) resulta em VERDADEIRO ; "A" > "B" ("A" está depois de "B" na ordem alfabética?) resulta em FALSO.

Lembrete:

Hierarquia das Operações Aritméticas:

1 ° () Parênteses

2 ° Exponenciação

3 ° Multiplicação, divisão (o que aparecer primeiro)

4 ° + ou – (o que aparecer primeiro)

Atividade:

1) Sabendo que $A=3$, $B=7$ e $C=4$, informe se as expressões abaixo são verdadeiras ou falsas

a) $(A+C) > B$

b) $B \geq (A + 2)$

c) $C = (B - A)$

d) $(B + A) \leq C$

e) $(C+A) > B$

2) Sabendo que $A=5$, $B=4$ e $C=3$ e $D=6$, informe se as expressões abaixo são verdadeiras ou falsas

a) $(A > C)$ **AND** $(C \leq D)$

b) $(A+B) > 10$ **OR** $(A+B) = (C+D)$

c) $(A \geq C)$ **AND** $(D \geq C)$

Teste de Mesa

Após desenvolver um algoritmo ele deverá sempre ser testado. Este teste é chamado de **TESTE DE MESA**, que significa, seguir as instruções do algoritmo de maneira precisa para verificar se o procedimento utilizado está correto ou não. O teste de mesa seria a simulação da execução do programa. Esse teste visa identificar a existência de erros de lógica.

O Visualg possui um campo que realiza o teste de mesa automaticamente ao solicitar a execução passo a passo. Ao digitar o valor correspondente ao número o valor da variável irá modificar de 0 para o número 10 que foi digitado pelo usuário. Conforme ilustrado na figura 2.

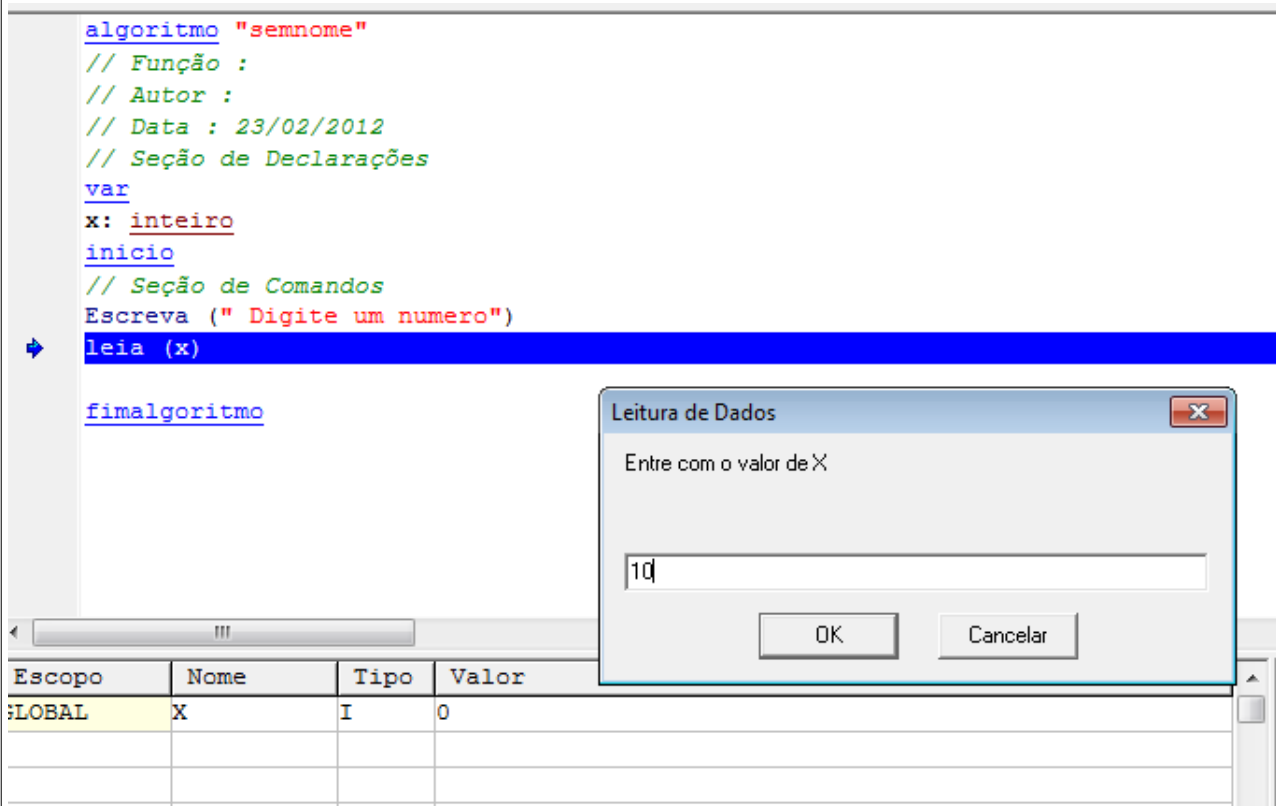


Figura 2- Campo do teste de Mesa

Comandos Básicos

Cada linguagem de programação contém um conjunto de comandos previamente definidos para a estruturação de um programa.

- Comando de Entrada: O comando de entrada de dados obtém um ou mais valores e coloca-os em variáveis.

Sintaxe: `leia identificador [identificador..]`

Ex.: `Leia n`

No visualg o programa ficaria conforme figura 3.

```

algoritmo "semnome"
// Função :
// Autor :
// Data : 14/02/2012
// Seção de Declarações
var
n:inteiro
inicio
// Seção de Comandos
escreva ("digite um numero")
leia (n)
fimalgoritmo

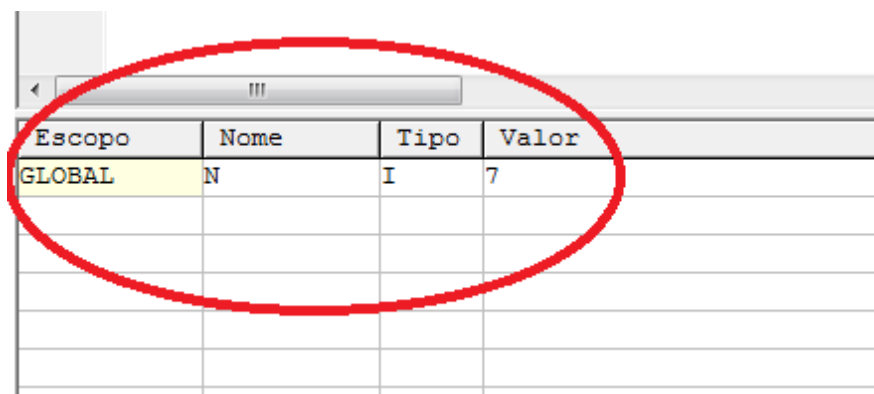
```

Figura 3- Exemplo de programa usando comando de entrada

Atenção

Ao fazer a leitura de um dado (uso do comando leia), o computador irá armazenar na variável que foi declarada para tal.

Observe que ao solicitar um número ao usuário, quando digitado o valor vai ser armazenado na área destinada a variável, conforme figura 4.



Escopo	Nome	Tipo	Valor
GLOBAL	N	I	7

Figura 4 – Espaço destinado a alocação das variáveis

- Comando de Saída: O comando de saída de dados exibe o valor de uma expressão.

Sintaxe: Escreva expressão[expressao...]

Ex.: Escreva media

Escreva (" reprovado")

O comando ESCRIVA pode ser usado para mostrar uma variável ou uma expressão. Quando for mostrar uma expressão esta deverá estar entre parentese, conforme a figura 5.

```
algoritmo "semnome"
// Função :
// Autor :
// Data : 14/02/2012
// Seção de Declarações
var
n:inteiro
inicio
// Seção de Comandos
escreva ("digite um numero")
leia (n)
escreva ("O NUMERO DIGITADO É", n)
fimAlgoritmo
```

Figura 5 – Utilização do comando Escreva no Visualg

- Comando de atribuição: O comando de atribuição armazena em uma variável o resultado da avaliação de uma expressão.

Sintaxe: identificador ← expressão

Ex.: media ← (n1+n2+n3)/3

```

algoritmo "semnome"
// Função :
// Autor :
// Data : 14/02/2012
// Seção de Declarações
var
n1,n2:inteiro
media: inteiro
inicio
// Seção de Comandos
escreva ("digite dois numeros")
leia (n1, n2)
media <- (n1 + n2)/2

finalgoritmo

```

Figura 6– Programa em visualg que calcula a média aritmética de dois números

Observe que o programa utiliza comando de entrada para receber os valores correspondentes a n1 e n2. Executa o cálculo da média através do comando de atribuição.(Figura 6)

– Estruturas de Controle: Sequencia e Seleção

1- Sequencia

Estrutura básica de qualquer algoritmo e determina a execução de passos consecutivos.

A sequencia estabelece uma ordenação temporal da execução.

2- Seleção

Comandos de Decisão : Conduzem a estruturas de programas que não são totalmente seqüenciais.

Com as instruções de SALTO ou DESVIO pode-se fazer com que o programa proceda de uma ou outra maneira, de acordo com as decisões lógicas tomadas em função dos dados ou resultados anteriores.

Principais estruturas de decisão

“**Se Então**” - Seleção Simples

Se determinada condição for satisfeita pelo comando SE/IF então execute determinado comando.

```
algoritmo "Condicional"
// Função :
// Autor :
// Data : 16/02/2012
// Seção de Declarações
var
    sexo: caracter
início
// Seção de Comandos

Escreva (" Digite m ou f par sexo")
Leia (sexo)
se sexo="f" entao
    escreva ("feminino")
fimse

finalgoritmo
```

Figura 7 – Exemplo de Programa utilizando a seleção simples

“**Se então Senão**” - Seleção Composta

caso a condição seja “verdadeira” o comando da condição será executado, caso contrário o comando da condição “falsa” será executado

```
algoritmo "Condicional"
// Função :
// Autor :
// Data : 16/02/2012
// Seção de Declarações
var
    sexo: caracter
início
// Seção de Comandos

Escreva (" Digite m ou f par sexo")
Leia (sexo)
se sexo="f" entao
    escreva ("feminino")
senao
    escreva (" masculino")
fimse

finalgoritmo
```

Figura 8- Exemplo de um programa usando a seleção composta

“**Escolha Caso**” - seleção Multipla

Utilizada para testar, na condição, uma única expressão, que produz um resultado, ou, então, o valor de uma variável, em que está armazenado um determinado conteúdo. Compara-se o resultado obtido no teste com os valores fornecidos em cada cláusula “Caso”. (Figura 9)

```
algoritmo "Condicional"
// Função :
// Autor :
// Data : 16/02/2012
// Seção de Declarações
var
    sexo: caracter
inicio
// Seção de Comandos

Escreva (" Digite m ou f para sexo")
Leia (sexo)
escolha sexo
caso "f"
    escreva ("feminino")
caso "m"
    escreva ("masculino")
fimescolha
finalgoritmo
```

Figura 9 – Exemplo de programa usando a seleção múltipla

Atividades

1)Desenvolva um programa que:

Leia 4 (quatro) números;

Calcule o quadrado de cada um;

Se o valor resultante do quadrado do terceiro for ≥ 1000 , mostre-o e finalize;

Caso contrário, mostre os valores lidos e seus respectivos quadrados.

2)Elabore um programa que dada a idade de um nadador classifique-o em uma das seguintes categorias:

Infantil A = 5 a 7 anos

Infantil B = 8 a 11 anos

Juvenil A = 12 a 13 anos

Juvenil B = 14 a 17 anos

Adultos = Maiores de 18 anos

Estrutura de Repetição

Utilizado quando desejamos que um determinado conjunto de instruções ou comandos sejam executados um número definido ou indefinido de vezes, ou enquanto um determinado estado de coisas prevalecer ou até que seja alcançado.

Principais estruturas de repetição do Visualg

Repetição com condição testada no início

O bloco de operações será executado enquanto a condição x for verdadeira

O teste da condição será sempre realizado antes de qualquer operação.

Enquanto a condição for verdadeira o processo se repete

Sintaxe: Enquanto < x > faça

Fimenquanto

```
algoritmo "Repetição"
// Função :
// Autor :
// Data : 17/02/2012
// Seção de Declarações
var
n: inteiro
inicio
// Seção de Comandos
// programa que mostra Vamos praticar! 10 vezes
n<- 0
enquanto n<10 faça
escreva (" VAMOS PRATICAR!")
n<- n+1
fimenquanto
|
finalgoritmo
```

Figura 10- Exemplo de programa usando Enquanto

Obs.: A variável *n* utilizada na figura 10, é utilizada como um contador, por isso é necessário que ele seja acrescido de um para que possa ser dado o giro no laço de repetição, caso contrário ele entrará em *loop* (repetição infinita).

– **Repetição testada no final do bloco**

O bloco de operações será executado enquanto a condição *x* for verdadeira

O teste da condição será sempre realizado ao final do bloco de operações.

Repete até que a condição seja satisfeita.

Sintaxe: Repita ...até que <x>

```
algoritmo "Repetição"
// Função :
// Autor :
// Data : 17/02/2012
// Seção de Declarações
var
n: inteiro
inicio
// Seção de Comandos
// programa que mostra Vamos praticar! 10 vezes
n<- 0
repita
escreva (" VAMOS PRATICAR!")
n<-n+1
ate n>10

finalgoritmo
```

Figura 11- Exemplo de programa usando Repita

Atenção

Os comandos Enquanto e Repita realizam a mesma função porém o que muda é o teste da condição. No enquanto o teste é realizado no início e no repita no final.

-Repetição com número determinado

Repete a execução do bloco um número definido de vezes, pois ela possui limites fixos

Sintaxe: Para ... de.. Até ...

Fimpara

```
algoritmo "Repetição"
// Função :
// Autor :
// Data : 17/02/2012
// Seção de Declarações
var
i: inteiro
inicio
// Seção de Comandos
// programa que mostra Vamos praticar! 10 vezes

para i de 1 ate 10 faca
escreva (" VAMOS PRATICAR!")
fimpara

finalgoritmo
```

Figura 12 – Exemplo de repetição usando para

Atividades

- 1) Escreva um programa que pergunte nome, idade e sexo Mostre sua idade daqui a 50 anos.
- 2) Determine o maior e o menor valor de um conjunto de números inteiros positivos. Considere que o conjunto de dados de entrada termina quando é fornecido o número 5.
- 3) Faça um algoritmo que conte de 1 a 100 e a cada múltiplo de 10 emita uma mensagem: "Múltiplo de 10".
- 4) Elabore um programa que leia uma massa de dados contendo SEXO, IDADE e ESTADO CIVIL (Casado/ Solteiro/Divorciado/ Outros) de um grupo de 100 pessoas e determine, ao final:
 - a) Média de idade das mulheres
 - b) Estado civil mais relevante entre os entrevistados.
- 5) Faça um algoritmo que leia dois números e identifique se são iguais ou diferentes. Caso eles sejam iguais imprima uma mensagem dizendo que eles são iguais. Caso sejam diferentes, informe qual número é o maior, e uma mensagem que são diferentes.
- 6) Escrever um algoritmo que leia três valores inteiros distintos e os escreva em ordem crescente
- 7) Escrever um algoritmo que leia três valores inteiros e verifique se eles podem ser os lados de um triângulo. Se forem, informar qual o tipo de triângulo que eles formam: equilátero, isóscele ou escaleno. Propriedade: o comprimento de cada lado de um triângulo é menor do que a soma dos comprimentos dos outros dois lados.

- 8) Escreva um programa que leia um número e se for par imprima “ número par”, se for ímpar imprimir “número ímpar”.
- 9) Escreva um programa que pergunte três nomes diferentes e os imprima em ordem alfabética.
- 10) Um autódromo necessita de um programa para verificar qual o carro mais rápido em uma competição. Considerando que dois carros participam da competição, o programa pergunta qual o comprimento da pista(em metros) e quanto tempo (em segundos) cada um dos carros levou para percorrer a pista, e informa a velocidade do carro mais rápido. Considere que em uma corrida não existe dois carros com tempos de percurso exatamente iguais.

Principais Funções

Todo programa vem com um conjunto de funções para facilitar a vida do programador, é como se fosse uma caixa preta que resolve o cálculo proposto. Estas funções realizam os cálculos aritméticos, trigonométricos e de manipulação e conversão de dados mais comuns; assim, o programador não tem que reinventar a roda a cada programa que faz. A este grupo de funções dá-se às vezes o nome de biblioteca.

As funções podem ser atribuídas a uma variável.

Funções numéricas, algébricas e trigonométricas

Abs(expressão) - Retorna o valor absoluto de uma expressão do tipo inteiro ou real. Equivale a $| \text{expressão} |$ na álgebra.

ArcCos(expressão) - Retorna o ângulo (em radianos) cujo co-seno é representado por expressão.

ArcSen(expressão) - Retorna o ângulo (em radianos) cujo seno é representado por expressão.

ArcTan(expressão) - Retorna o ângulo (em radianos) cuja tangente é representada por expressão.

Cos(expressão) - Retorna o co-seno do ângulo (em radianos) representado por expressão.

CoTan(expressão) - Retorna a co-tangente do ângulo (em radianos) representado por expressão.

Exp(base, expoente) - Retorna o valor de base elevado a expoente, sendo ambos expressões do tipo real.

GraupRad(expressão) - Retorna o valor em radianos correspondente ao valor em graus representado por expressão.

Int(expressão) - Retorna a parte inteira do valor representado por expressão.

Log(expressão) - Retorna o logaritmo na base 10 do valor representado por expressão.

LogN(expressão) - Retorna o logaritmo neperiano (base e) do valor representado por expressão.

Pi - Retorna o valor 3.141592.

Quad(expressão) - Retorna quadrado do valor representado por expressão.

RadpGrau(expressão) - Retorna o valor em graus correspondente ao valor em radianos representado por expressão.

RaizQ(expressão) - Retorna a raiz quadrada do valor representado por expressão.

Rand - Retorna um número real gerado aleatoriamente, maior ou igual a zero e menor que um.

RandI(limite) - Retorna um número inteiro gerado aleatoriamente, maior ou igual a zero e menor que limite.

Sen(expressão) - Retorna o seno do ângulo (em radianos) representado por expressão.

Tan(expressão) - Retorna a tangente do ângulo (em radianos) representado por expressão.

Os valores que estão entre parênteses, representados pelas palavras como *expressão*, *base* e *expoente*, são os parâmetros, ou como dizem alguns autores, os argumentos que passamos para a função para que realize seus cálculos e retorne um outro, que usaremos no programa. Algumas funções, como Pi e Rand, não precisam de parâmetros, mas a maioria tem um ou mais. O valor dos parâmetros naturalmente altera o valor retornado pela função.

Funções para manipulação de cadeias de caracteres (strings)

Asc (s : character) : Retorna um inteiro com o código ASCII do primeiro caracter da expressão.

Carac (c : inteiro) : Retorna o caracter cujo código ASCII corresponde à expressão.

Caracpnum (c : character) : Retorna o inteiro ou real representado pela expressão. Corresponde a StrToInt() ou StrToFloat() do Delphi, Val() do Basic ou Clipper, etc.

Compr (c : character) : Retorna um inteiro contendo o comprimento (quantidade de caracteres) da expressão.

Copia (c : character ; p, n : inteiro) : Retorna um valor do tipo character contendo uma cópia parcial da expressão, a partir do caracter p, contendo n caracteres. Os caracteres são numerados da esquerda para a direita, começando de 1. Corresponde a Copy() do Delphi, Mid\$() do Basic ou Substr() do Clipper.

Maiusc (c : character) : Retorna um valor character contendo a expressão em maiúsculas.

Minusc (c : character) : Retorna um valor character contendo a expressão em minúsculas.

Numpcarac (n : inteiro ou real) : Retorna um valor character contendo a representação de n como uma cadeia de caracteres. Corresponde a IntToStr() ou FloatToStr() do Delphi, Str() do Basic ou Clipper.

Pos (subc, c : character) : Retorna um inteiro que indica a posição em que a cadeia subc se encontra em c, ou zero se subc não estiver contida em c. Corresponde funcionalmente a Pos() do Delphi, Instr() do Basic ou At() do Clipper, embora a ordem dos parâmetros possa ser diferente em algumas destas linguagens.

Sugestão: Teste essas funções criando programas no Visualg, seja criativo e bom estudo !

Referências

ARAUJO, Everton Coimbra. **Algoritmos: Fundamento e Prática**. Florianópolis: Visual Books, 2007.

FARRER, Harry et all. **Algoritmos Estruturados**. Rio de Janeiro: LTC, 2011.

VENANCIO, C.F.; **Desenvolvimento de Algoritmos**; Editora Erica.